1 *El código que se muestra seguidamente es una implementación de una primitiva de adquisición de un cerrojo con espera activa* (spin-lock) *empleando la instrucción atómica* `lock.xchg .Ri, /dir`. *Indique cómo se comporta en cuanto número de fallos por invalidación (*invalidation misses*) para máquinas UMA con cachés privadas y mecanismo de coherencia por Invalidación. Reescriba la implementación propuesta de manera que se reduzca este número de fallos.*

```
          LD .R0, #D'01
intento:  lock.xchg .R0, /dir_cerrojo
          bnz $intento
          ret
```

SOLUCIÓN

Si se analiza la implementación propuesta, se observa que trata de reproducir el comportamiento de una instrucción de `test&set` mediante la instrucción atómica `lock.xchg`, tal y como se desarrolló en una de las sesiones de ejercicios de la asignatura. También presupone (dependería de la arquitectura particular que se usase) que la ejecución de la instrucción afecta a los flags aritméticos (según el valor que retorna el registro Ri, en este caso mientras no devuelva el valor cero).

En cada vuelta (*spin*) al bucle se escribe un uno en la dirección del cerrojo, tal y como lo haría un `test&set`, cuyo comportamiento se trata de emular, y dando lugar a la invalidación de bloque. Consiguientemente, se produciría un fallo por invalidación en cada una de las restantes copias del bloque, presentes en las cachés de los otros procesadores que estuviesen compitiendo por el acceso al cerrojo (que, a su vez, al realizar la escritura mediante el `lock.xchg`, darían lugar a una nueva invalidación de las demás copias).

Este comportamiento se atenuaría significativamente si primero se comprobase que el cerrojo está abierto (y sin modificar el bloque correspondiente ni dando lugar a los consiguientes invalidación y fallos), y luego se tratase de escribir un uno en él, en una estructura que se suele denominar `test&test&set` y estudiada también en la asignatura. Una posible codificación sería la siguiente reescritura del código propuesto en el enunciado:

```
test:     ld .R0, /dir_cerrojo
          cmp .R0, #D'00
          bnz $test
          ld .R0, #D'01
          lock.xchg .R0, /dir_cerrojo
          bnz $test
          ret
```

2 *Un programa ejecuta un total de* $120 \cdot 10^8$ *instrucciones. De ellas, el 75 % se ejecutan en tres ciclos de reloj y el resto lo hace en cinco ciclos. Para obtener el tiempo de ejecución de este programa se utiliza la orden* `time` *del sistema operativo, que proporciona la siguiente información:*

```
real 0m84s
user 0m34s
sys  0m1s
```

Calcule el número medio de ciclos por instrucción (CPI) en el programa, la frecuencia del procesador y la productividad expresada en MIPS.

SOLUCIÓN

Número medio de ciclos por instrucción:

$CPI = 0,75.3 + 0,25.5 = 3,5$ *ciclos/instruccion*

A partir de la información proporcionada por la orden *time*, deducimos que el tiempo de ejecución es:

$Tejecucion = user + sys = (34 + 1)s = 35s$

Sabiendo que:

$Tejecucion = Tciclo.N^{o}instrucciones.CPI$

Calculamos la frecuencia del procesador del siguiente modo:

$frecuencia = \frac{N^o instrucciones.CPI}{Tejecucion} = \frac{120.10^8.3,5}{35} ciclos/s = 1,2GHz$

Productividad expresada en MIPS:

$MIPS = \frac{N^o instrucciones}{Tejecucion.10^6} = \frac{120.10^8}{35} = 342,85$

3 *Se dispone de un computador A con un procesador RISC que incorpora arquitectura Harvard en el nivel de memoria caché, y con un pipeline de instrucciones de 5 etapas:*

E1: Búsqueda de instrucción e incremento del PC.

E2: Decodificación y lectura de registros.

E3: Ejecución, cálculo de direcciones en las instrucciones `ld` *y* `st`*, comprobación de condición en los saltos condicionales y cálculo de dirección de salto.*

E4: Acceso a memoria de datos.

E5: Escritura en registros.

*El procesador no dispone de mecanismos de adelantamiento (*forwarding*), e introduce los ciclos de espera necesarios para resolver las posibles dependencias de datos y de control. Además, el ciclo de acceso al banco de registros está dividido en 2 subciclos (en el primero se escribe y en el segundo se lee).*

Se quiere sustituir este computador por otro, B, cuyo procesador tiene las mismas características que el anterior, salvo que dispone de todo tipo de mecanismos de adelantamiento e incorpora predicción dinámica de salto de 2 bits. Dicha predicción se realiza en la etapa E1 del pipeline.

Para evaluar la ganancia real que se obtendría con computador B se emplea, entre otros, el siguiente fragmento de código:

```
(1)  buc:  ld r5, 0(r10)       ; for (i=0; i<500; i++)
(2)        add r15, r5, r5     ;  for (j=0; j<100; j++)
(3)        ld r6, 0(r11)       ;    a[i][j] = 2*a[i][j]+b[i][j];
(4)        add r5, r5, r6
(5)        st r5, 0(r10)
(6)        add r1, r1, 1
(7)        add r10, r10, 4
(8)        add r11, r11, 4
(9)        sub r3, r1, r4
(10)       bnz r3, buc
(11)       add r1, r0, r0
(12)       add r2, r2, 1
(13)       sub r3, r2, r5
(14)       bnz r3, buc
```

a) *Indique razonadamente las dependencias de datos y de control que generan parones, así como el número de ciclos de espera que debe introducir cada uno de los procesadores para resolverlas. En el caso de las dependencias de control para el computador B indique el número de ciclos de espera que se introducen tanto en el caso de acierto como en el de fallo de predicción.*

b) *Calcule el tiempo de ejecución del fragmento de código en ambos casos, distinguiendo claramente entre el tiempo empleado en la ejecución de instrucciones y el correspondiente a los ciclos de parada de los distintos tipos de dependencias. Para la ejecución en el computador B suponga que la primera vez que se ejecuta cada instrucción de salto los bits de predicción tienen el valor 11.*

c) *Calcule la ganancia que se obtendría utilizando el computador B.*

d) *Cuantifique cual de las dos diferencias entre el computador A y el B influye más en el incremento de rendimiento que se consigue con B.*

SOLUCIÓN

a) A continuación se indican las instrucciones entre las que existen dependencias de datos, todas del tipo RAW, así como los ciclos de espera introducidos en cada procesador.

Instrucciones	Ciclos de espera en A	Ciclos de espera en B
(1) y (2)	2	1 (adelantamiento MEM-ALU)
(3) y (4)	2	1 (adelantamiento MEM-ALU)
(4) y (5)	2	0 (adelantamiento ALU-MEM)
(9) y (10)	2	0 (adelantamiento ALU-ALU)
(12) y (13)	2	0 " "
(13) y (14)	2	0 " "

En A las dependencias entre dos instrucciones consecutivas generan 2 ciclos de espera debido a que la instrucción que genera el dato escribe en el banco de registros en el primer subciclo de la etapa E5 y la instrucción que utiliza dicho dato lo debe leer en el segundo subciclo de la etapa E2.

En el caso de B, el uso de adelantamiento elimina los ciclos de espera en todos los casos salvo cuando la instrucción que genera el dato es una instrucción `ld`. En este caso, dicha instrucción tiene el dato disponible al finalizar la etapa E4 y la instrucción que utiliza dicho dato lo hace en la etapa E3, por lo que hay que introducir un ciclo de espera entre ambas para el funcionamiento correcto de la segunda instrucción.

En cuanto a las dependencias de control, éstas se producen debido a las dos instrucciones de salto (10) y (14). En el caso de A, introducirá 2 ciclos de espera detrás de cada instrucción de salto ejecutada, puesto que la condición y la dirección de salto se calculan en la etapa E3. En el caso de B, si hay acierto en la predicción no se introduce ningún ciclo de espera ya que la predicción se realiza en E1, mientras que en el caso de fallo se introducen 2 ciclos de espera puesto que la comprobación de la condición y por lo tanto de la predicción se realiza en E3.

b) Teniendo en cuenta los resultados del apartado anterior, el tiempo de ejecución en cada uno de los computadores es el siguiente:

Tej(A) = 4 ciclos de llenado del pipeline +

(500x100x10 + 500x4) ciclos de ejecución de instrucciones +

(500x100x8 + 500x4) ciclos de espera por dependencias RAW +

(500x100x2 + 500x2) ciclos de espera por dependencias de control =

= 4 + 502.000 + 402.000 + 101.000 = **1.005.004 ciclos**

Tej(B) = 4 ciclos de llenado del pipeline +

(500x100x10 + 500x4) ciclos de ejecución de instrucciones +

(500x100x2) ciclos de espera por dependencias RAW +

(500x2 + 2) ciclos de espera por dependencias de control =

= 4 + 502.000 + 100.000 + 1.002 = **603.006 ciclos**

En el caso de B, los 1.002 ciclos de espera por dependencias de control se deben a los 500 fallos de predicción que se producen en la ejecución de la instrucción (10), cada vez que se sale del bucle j, y al fallo de predicción de (14) cuando se sale del bucle i.

c) La ganancia que se obtendría con el computador B se calcula dividiendo el tiempo de ejecución obtenido con el computador A entre el obtenido con el computador B, que para el código del enunciado sería:

1.005.004/603.006 = **1,67**

d) Una forma de cuantificar cual de las dos diferencias entre A y B influyen más en el incremento de rendimiento que se consigue con B es la que se describe a continuación.

- Diferencia de tiempos de ejecucion entre A y B:

  1.005.004 - 603.006 = 401.998 ciclos

- Diferencia entre el numero de ciclos debidos a dependencias RAW entre A y B:

  (500x100x8 + 500x4) - (500x100x2) = 302.000 ciclos

que representa el **75,12 %** de la reduccion ((302.000/401.998)x100)

- Diferencia entre el numero de ciclos debidos a dependencias de control entre A y B:

  (500x100x2 + 500x2) - (500x2 + 2) = 99.998 ciclos

  que representa el **24,88 %** de la reduccion

Como se puede comprobar por los cálculos realizados, el mecanismo de adelantamiento tiene una mayor influencia que la predicción de salto en el incremento de rendimiento obtenido con el computador B.

4 *Sea un computador con memoria virtual paginada que utiliza dos niveles de tablas de páginas, direcciones virtuales de 36 bits y direcciones físicas de 32 bits. Para realizar la traducción utiliza también dos TLBs, una para instrucciones y otra para datos, ambas con tiempo de acceso de 2 ns, 6 entradas, totalmente asociativas y con política de reemplazo LRU. Las páginas son de 4 KB, y todas las tablas necesarias para la traducción tienen 2^{12} entradas. En este computador se está ejecutando un programa de cuyo código se ha extraído, para ser analizado, el siguiente fragmento:*

```
/*  Fragmento de código para analizar: */
max = 128 * 128;        /* = 16384 */
for (j=0; j<max; j++)
     y[j] = 2 * x[j];
verif(y);
for (j=0; j<max; j++)
     x[j] = y[j] / 3;
```

Los vectores `x` *e* `y`*, cuyos elementos ocupan un byte, comienzan a partir de las siguientes direcciones virtuales:*

`Dv(x) = 002003000H, Dv(y) = 003001000H`

a) *Indique el formato de las direcciones virtuales y describa cómo se han de interpretar las direcciones asociadas a los vectores* `x` *e* `y`*.*

b) *Suponga, además, que en un momento dado de la ejecución del código proporcionado, toda la información correspondiente a los vectores* `x` *e* `y` *se encuentra en direcciones contiguas de memoria física que comienzan en las direcciones siguientes:*

`Df(x) = 00020000H, Df(y) = 00038000H`

Suponiendo que todas las tablas de traducción se encuentran en memoria principal y que la TLB de datos no contiene información relativa a los datos de dicho fragmento de código, indique y justifique el contenido de las entradas de las distintas tablas de traducción que se utilizan para encontrar la dirección física de `x` *e* `y`*. Para ello:*

- *Tenga en cuenta que el tamaño de los vectores es el determinado por la constante* `max` *del código anterior.*
- *Asigne las direcciones arbitrarias que considere oportuno a las tablas de segundo nivel necesarias para realizar esta traducción, indicando claramente cuál es esta asignación.*

c) *Suponiendo que las variables* `i`*,* `j` *y* `max` *están asignadas a registros, y que la función* `verif` *no realiza accesos a memoria para leer o escribir datos, calcule el número de fallos que se producen en la TLB de datos al ejecutar este fragmento de código. Especifique en qué parte(s) del código se producen estos fallos. Justifique si otra política de reemplazo diferente de LRU podría mejorar los resultados en cuanto a la tasa de aciertos de esta TLB.*

d) *De todas las entradas de la TLB de datos, indique cuántas se utilizan y cuál será el contenido (etiquetas e información de traducción) de las entradas correspondientes a la primera página de* `x` *y a la última de* `y`*.*

e) *Calcule el tiempo total empleado en el acceso a los datos. Para ello suponga que los vectores* `x` *e* `y` *no pasan por la memoria cache, y que el tiempo de acceso de la memoria principal es de 40 ns. Calcule qué porcentaje de este tiempo corresponde a la traducción y cuál al acceso a los datos.*

f) *Suponga que durante la ejecución del fragmento de código a analizar se produce un cambio de contexto y se pasa a ejecutar otro programa que ejecuta un fragmento de código exactamente igual, y en el que* `x` *e* `y` *tienen asignadas además las mismas direcciones virtuales. Identifique cuántos fallos se producirán en la TLB de datos en este caso. Justifique si las direcciones físicas asignadas a* `x` *e* `y` *serán también coincidentes para los dos procesos.*

SOLUCIÓN

a) Según establece el enunciado, todas las tablas de traducción, tanto de primer nivel como de segundo, tienen 2^{12} entradas. Debido a ello, cada uno de los campos `zona` y `página` de la dirección virtual ocupa 12 bits. Los 12 bits menos significativos se utilizan para seleccionar el byte dentro de la página, ya que cada una de éstas tiene un tamaño de 4 KB.

El formato de las direcciones virtuales es el que se muestra a continuación:

Zona	Página	Byte (desplazamiento)
12 bits	12 bits	12 bits

La interpretación de las direcciones virtuales de comienzo de `x` e `y` según este formato es:

Dv(x) = `002003000H`. Separando estos dígitos hexadecimales en bloques de 12 bits equivale a:

`002 003 000 Hex` y significa Zona:2, PáginaVirtual:3, Desplazamiento:0.

Dv(y) = `00300100H`. Separando estos dígitos hexadecimales en bloques de 12 bits equivale a:

`003 001 000 Hex` y significa Zona:3, PáginaVirtual:1, Desplazamiento:0.

b) El tamaño de los vectores utilizados, `x` e `y`, es de $128 \times 128 = 16384 = 2^{14}$ *Bytes*, o, lo que es igual, 4 páginas de memoria. En consecuencia, las tablas de traducción contendrán información para traducir 4 páginas de `x` y 4 páginas de `y`.

Necesariamente se tendrá en memoria la tabla de traducción de primer nivel `TP1`, que según se ha visto en el apartado anterior, deberá indicar la dirección de la tabla 2 de segundo nivel (`TP2_2`) para `x` y de la tabla 3 de segundo nivel (`TP2_3`) para `y`. Por otra parte, en la tabla `TP2_2` estará la información necesaria para traducir cuatro páginas consecutivas a partir de la 3, para el vector `x`. En la tabla `TP2_3` se encontrará la información necesaria para traducir cuatro páginas consecutivas a partir de la 1, para el vector `y`. Asignando direcciones arbitrarias a la ubicación de las tablas `TP2_2` y `TP2_3` tal como sugiere el enunciado y teniendo en cuenta las direcciones físicas asociadas a los vectores `x` e `y`, se tendría la situación representada en la figura 1.

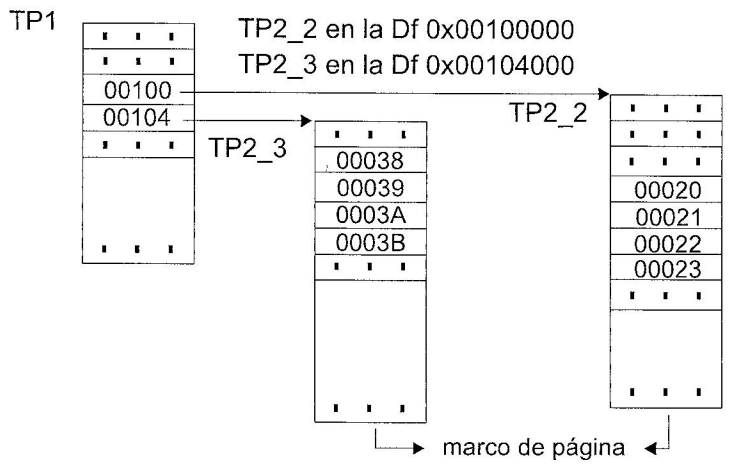

Figura 1. Contenido de las tablas de traducción.

c) Se produce un fallo de TLB la primera vez que se hace referencia a cada página. Se tendrá en cuenta que `x` e `y` ocupan cuatro páginas y que el código proporcionado realiza un acceso de lectura a `x[j]` y uno de escritura a `y[j]` en cada iteración del primer bucle.

La traza de ejecución del programa es tal que se recorren los dos vectores de forma secuencial en ambos bucles, accediendo a sus sucesivas palabras y, por lo tanto, a sus páginas consecutivas. En consecuencia, se producen los dos primeros fallos en el primer acceso a `x[0]` (zona 2, página 3) y el primero a `y[0]` (zona 3, página 1). Durante el recorrido de `x` e `y`, se irán produciendo fallos en los accesos a las páginas siguientes, llegando al final del primer bucle con 8 fallos de TLB (las cuatro páginas de cada vector) y quedando en TLB, debido a la política de reemplazo LRU, la traducción de las tres últimas páginas de `x` y las tres últimas de `y`.

Al entrar en el segundo bucle, la situación será prácticamente idéntica, ya que no se dispondrá inicialmente de la traducción de la primera página de `x` e `y`. De este modo, al actualizar la TLB se perderá la información

de traducción de la segunda página de cada vector (por LRU), lo que provocará un nuevo par de fallos de traducción, que se propagará hasta completar el segundo bucle con otro conjunto de 8 fallos.

En total se producen 16 fallos de TLB, ocho en cada uno de los bucles.

El uso de la política de reemplazo LRU hace que se produzca un nuevo fallo cada vez que se accede a una nueva página de cualquiera de los vectores utilizados en el fragmento de código de este ejemplo, por lo que otras políticas, por ejemplo, la aleatoria, podría lograr fácilmente una mejora en los resultados.

d) Tal como se desprende de lo expresado en el apartado anterior, se utilizan todas las entradas de la TLB de datos. El contenido estará formado por la etiqueta y la información de traducción. La etiqueta contendrá, al menos, la parte que se traduce de la dirección virtual. Podría contener también una identificación del proceso para el que se ha realizado la traducción. La información de traducción estará formada por la parte de la dirección física que surge de la traducción (no incluye por tanto los bits de desplazamiento dentro del marco de página), además de los permisos de acceso y otros bits para implementar las políticas de trabajo de la memoria virtual. Teniendo en cuenta únicamente la parte básica de la información mencionada, las entradas correspondientes a la primera página de x y la última de y podrían ser:

	PV	"MP ... prot. ..."
Primera página de x:	002003	00020 | ... permisos ...
Última página de y:	003004	0003B | ... permisos ...

e) Para calcular el tiempo total empleado en el acceso a datos se ha de tener en cuenta que la información utilizada no pasa por la memoria caché y las escrituras tardan el mismo tiempo que las lecturas. El número total de accesos a datos, contabilizando ambos bucles, es de:

$$2\ bucles \times 2 \times 128 \times 128 = 65.536\ accesos$$

De ellos, solo 16 se realizan con fallo de traducción en TLB (8 en cada bucle). Todos los accesos a datos, tanto con fallo como con acierto en TLB requieren un acceso a la TLB para hacer la traducción o detectar el fallo. La traducción necesaria para los 16 accesos con fallo de TLB requiere acceder a las tablas de traducción de primer y segundo nivel (es decir, dos accesos a memoria principal). Todos los accesos a datos requieren un acceso adicional a memoria para leer o escribir la información. En resumen, el tiempo invertido en los accesos a datos es:

$$t_{trad} = 65.536 \times 2\ ns + 16 \times (2 \times 40)\ ns = 132.352\ ns$$
$$t_{total} = t_{trad} + 65.536 \times 40\ ns = 2.753.792\ ns$$

En consecuencia, la proporción de tiempo dedicado a la traducción será de 132.352/2.753.792 = 0,048 (4,8 %) y la dedicada a realizar los accesos a memoria para leer o escribir, será de 65.536 × 40/2.753.792 = 0,952 (95,2 %).

f) Independientemente de que la TLB tenga o no un campo identificador del proceso, se producirán también 16 fallos cada vez que se cambie de proceso, ya que se hace referencia a 8 páginas distintas en cada bucle y solo hay capacidad para traducir 6. El hecho de usar las mismas direcciones virtuales es indiferente.

Las direcciones físicas serán necesariamente distintas para los diferentes procesos ya que, al contrario que el espacio virtual, el espacio físico está compartido entre todos ellos.